SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Evaluating learning algorithms: An efficient way/Efficient ways to find Regular Inductive Statements

Van Tu Nguyen

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Evaluating learning algorithms: An efficient way/Efficient ways to find Regular Inductive Statements

Titel der Abschlussarbeit

Author:	Van Tu Nguyen
Supervisor:	Prof. Dr. Dr. h. c. Javier Esparza
Advisor:	Dr. Christoph Welzel-Mohr
Submission Date:	Submission date

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, Submission date

Van Tu Nguyen

Abstract

Acknowledgments

I could not have undertaken this journey without supporting of Christoph Welzel-Mohr. I want to thank Christoph Welzel-Mohr for guiding me during the thesis.

I'd like to acknowledge to Chair of Theoretical Computer Science for providing the foundational knowledge necessary for me to finish this thesis.

Last but not least, thanks also go to my family, which was both supportive and patient.

Danke!
Thank you!
Cam on!

Contents

1 Introduction

As software systems grow in size and permeate more and more areas of our lives. Individuals and organizations use the majority of software in their systems. Thus the reliability and stability of the software testing are of major importance. Simulation and testing can detect bugs but not prove their absence. Such reactive systems, when no function is being computed, termination is usually undesirable. For this reason, we are interested in *property checking* or *model checking*. It has found a wide range of applications spanning from adaptive model checking.

We here consider the verification of safety properties similar to the original Regular Model Checking framework, where a program is represented using symbols and finite automata. To be more specific, our goal is to confirm that a given program cannot execute in a way that starts from a set of initial configurations ($\mathcal{I}$) and leads to a group of dedicated bad configurations ($\mathcal{B}$). These bad configurations represent conditions that should not happen during the program's execution. However, this is an undecidable question in general and tools for Regular Model Checking are necessarily incomplete. A solution to this problem was proposed in [CES09], which utilizes *inductive statements* to ensure that no undesired configuration can be reached from any initial configuration. This means that for every pair of initial and undesired configurations, there is at least one inductive statement that is satisfied by the initial configuration but not by the undesired one. By doing this, it can be concluded that no undesired configuration can be reached.

Over the past decade, there has been a significant increase in the study of automata learning. This field has produced numerous successful applications, such as pattern and natural language recognition, computational biology, data mining, robotics, automatic verification, and even the analysis of music. One can use autoamta learning to acquire a set of inductive statements that are powerful enough to establish a given safety property. The language of these inductive statements serves as proof of the property's correctness. The purpose of this thesis is to collect and analyze empirical data on the performance of learning algorithms such as L*, NL*, Kearns-Vazirani, Rivest-Schapire.

Structure of the thesis

In this thesis, we begin with Chapter 3 by fixing notations and definitions used throughout the thesis. In Chapter 4, we will thoroughly explain the *Regular transition system* and *Inductive statements* as an approach for checking the safety properties of *model checking*. Subsequently, Chapter 5 gives a general introduction to active learning algorithm and their oracles. Furthermore, we will introduce some active learning algorithms used for our experiments. Chapter 6 will investigate the C++-implemented programm to learn a set of inductive statements from a system configuation called *dodo*. The programm uses not only the *Angluin's algorithm* L^*, but also the NL^*, *Kearns-Vaziran* and *Rivest-Schapi*. After learning process, it visualizes the graphs that can evaluate the *efficiency* and *effectiveness* of these algorithms. Finally, we will summarize and assess our experment results in Chapter 7 and conclude the thesis with Chapter 8.

2 Literature Review

Inductive statements for regular transition system was introduced by Dr. Welzel-Mohr. Motivated from paper Inductive statements for regular transition system. The author used only the L* algorithm for learning the regular statements.

3 Preliminaries

In this chapter, we introduce some basic notions and definitioms that we use throughout this thesis.

Finite automata

We distinguish between deterministic and non-deterministic automata to recognize regular languages of finite words.

Definition 3.1: Deterministic finite automaton (DFA)

A DFA is a quintuple $\mathcal{M} = (Q, q_0, \Sigma, \delta, F)$ where Q is a finite set of states with a initial state $q_0 \in Q$. A set of input symbols called the alphabet Σ. A transition $\delta : Q \times \Sigma \rightarrow Q$ and a set of final states F. Let $w = a_1 a_2 ... a_n$ be a string over the alphabet Σ. The automaton $\mathcal{M}$ accepts w if a sequence of states, $r_0, r_1, ... r_n$ exist in Q:

- $r_0 = q_0$
- $r_{i+1} = \delta(r_i, a_{i+1}), for\ i = 0, ..., n-1$
- $r_n \in F$

Definition 3.2: Nondeterministic finite automaton (NFA)

A NFA is a quintuple $\mathcal{N} = (Q, q_0, \Sigma, \Delta, F)$ where Q, Σ and F are as for a DFA. Let $w = a_1 a_2 ... a_n$ be a string over the alphabet Σ. The automaton $\mathcal{N}$ accepts w if a sequence of states, $r_0, r_1, ... r_n$ exist in Q:

- $r_0 = q_0$
- $r_{i+1} \in \Delta(r_i, a_{i+1}), for\ i = 0, ..., n-1$

- $r_n \in F$

Token passing algorithm

We will provide a simple example to demonstrate how systems are modelled in *regular transition system*. The *token passing* system comprises a linear array of agents where the first agent holds a token, and in each step, the current agent can pass the token to its right neighbour. We choose to represent the agent that holds the token as the letter t and the agents that do not hold the token as the letter n.

4 Inductive statements for regular transition system

In the *Regular Model Checking* framework, program configurations are represented as finite words over a pre-determined alphabet Σ. The system comprises a series of starting configurations and the transitions are modelled as finite state transducers mapping configuations to configuations. In Section 4.1, we introduce *Regular transition system* (RTS) —an important framework for infinite state model-checking —to represent the behavior of a system.

4.1 Regular transition system

Essentially, a *regular transition system* represents a parameterized system $\mathcal{S}$. For example, $\mathcal{S}$ is the system of *token passing algorithm* with n is the number of agents. We call Σ is the set of alphabets of the system, which indicates the finite states of each agent. The sequentially alphabets of length n represents the current state of the system. In other words, one can understand that the first letter indicates the state of the first agent, and the second letter indicates the state of the second agent and so on. The states of each agent can be changed by following the rules of the system, which called the relations. Formally, we call that a *transducer* and define these relations in form of an NFA as follow:

Definition 4.1: Transducer

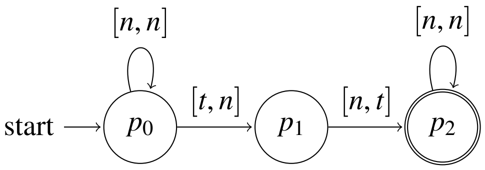
A Σ-Γ-*transducer* $\mathcal{T}$ *is an NFA* $\langle Q, Q_0, \Sigma \times \Gamma, \Delta, F \rangle$, *we denote a relation*

$$[[\mathcal{T}]] = \{\langle u_1 \dots u_n, v_1 \dots v_n \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Gamma^n \mid \langle u_1, v_1 \rangle \dots \langle u_n, v_n \rangle \in \mathcal{L}(\mathcal{T})\}$$

Note that this relationship is only applicable to words that have the same length. Extend this notation, we call

$$For\ v \in \Sigma^* : target_{\mathcal{T}}(v) = \{u \in \Gamma^* \mid \langle v, u \rangle \in [[\mathcal{T}]]\}$$

Automaton 4.1: Transducer Γ for Token passing



Intuitively, the Automaton 4.1 shows that the token will be passed from the left agent to the right agent, and once it reaches the end of the agents, no more transitions can be applied to the configuration. For example, from the configuration n n t n can be changed to n n n t, but n n n t can not be changed to any configurations. We capture the transitions of the *token passing* system via the language $[n,n]^*[t,n][n,t][n,n]^*$.

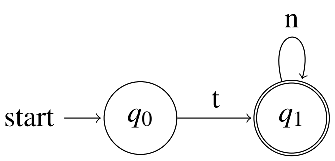
The initial configuations are actually the start states of some parameterized system. The system start with these initial states and can change it by follwing the transducer.

Definition 4.2: Regular transition system (RTS)

An RTS is a triple $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ where Σ is finite alphabet and $\mathcal{I}$ is an NFA, which represents initial configurations. $\mathcal{T}$ is a Σ-Σ-transducer of the system.

We denote with $\rightsquigarrow_{\mathcal{T}}$ the relation $[[\mathcal{T}]]$ and call a pair $\langle u, v \rangle \in \rightsquigarrow_{\mathcal{T}}$ a transition of $\mathcal{R}$. Moreover, let $\rightsquigarrow^*_{\mathcal{T}}$ denote the reflexive transitive closure of $\rightsquigarrow_{\mathcal{T}}$. We consider $w \in \Sigma^*$ *reachable* on $\mathcal{R}$ if there exist $u \in \mathcal{L}(\mathcal{I})$ with $u \rightsquigarrow^*_{\mathcal{T}} v$. Let $reach(\mathcal{R}) \subseteq \Sigma^*$ denote all reachable configuations.

Automaton 4.2: NFA $\mathcal{I}$ for Token passing



The set of initial configuations for *token passing* system is the language of NFA $\mathcal{I}$ or $\mathcal{L}(\mathcal{I}) = tn^*$. In other words, the first agent alway holds the toke while the following agents do not.

4.2 Inductive statements

Reachability problem The *RTS* $\mathcal{R}$ and the automaton $\mathcal{B}$ for the regular language that denotes the undesired configurations has been furnished now. The question is whether one can reach any undesired configuration in this transition system. Formally, we have to compute if $\mathit{reach}(\mathcal{R}) \bigcap \mathcal{L}(\mathcal{B}) = \emptyset$? For the reason that this reachability problem is undecidable in general we need a new approach that proves that no undesired configuation can be reached. We consider the question: "Is there a pair of configurations, v and u, such that u satisfies all the inductive statements satisfied by v?". Inductively, if u is reachable from v, then u will satisfied the inductive statements that v satisfied. By using inductive statements, it is possible to check whether an undesired configuration can be reached from an initial configuration. If there is an inductive statement satisfied by v but not u, then u cannot be reached from v.

Encoded statements We shall now proceed to examine the process of how the statements are encoded. We consider the statement pattern "in all configuations of a certain length m either agent i_1 is in state σ_1 or agent i_2 is in state σ_2 or . . . or agent i_k is in state σ_k". In general, the nesccessary information of any statement can be encoded as a function $f : \{1, \dots, m\} \rightarrow 2^{\Sigma}$, while the set of letters $f(i) \subseteq \Sigma$ corresponds to the states the i-th agent. In other words, each agent can be required by any states or not ($\emptyset$). For example, a similar statement could be "in all configurations of length 3 the first agent is in state p or the first agent is in state q". Using the above method, the example statement would be encoded as a function $\{1 \mapsto \{p, q\}, 2 \mapsto \emptyset, 3 \mapsto \emptyset\}$. We can simply this function to words $\{p, q\}\ \emptyset\ \emptyset$.

Without context, these words have no meaning or information. Therefore, we use *interpretation* to understand the statements that we encode. For example, how can we answer the question: "Does the words p q q satisfied the statement $\{p, q\}\ \emptyset\ \emptyset$?". One we can check whether $p \in \{p, q\}$, or $q \in \emptyset$. It depends on which interpretation do we use.

Definition 4.3: Interpretation

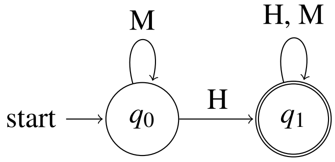
For any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, we call a pair $\langle \Gamma, \mathcal{V} \rangle$ an Γ-interpretation where Γ is a finite alphabet and $\mathcal{V}$ is a deterministic Σ-Σ-transducer. In the following, we denote $u \models I$ to indicate $\langle u, I \rangle \in [[\mathcal{V}]]$.

Concrete interpretations In this thesis, we will present three interpretations, and we will delve deeper into each of them for a better understanding.

Traps Let fix the size of the instance as n. We define any configuration $u_1 \dots u_n$ the set $\big(u\big) = \bigcup_{1 \leqslant i \leqslant n} \{\langle i, u_i \rangle\}$. For any statement $I_1 \dots I_n$ we define a set $\big(I\big) = \bigcup_{1 \leqslant i \leqslant n} \{i\} \times I_i$. The interpretation of a trap involves connecting a configuration u with a statement I if and only if $\big(u\big) \cap \big(I\big) \neq \emptyset$. Once a configuration has a value in the inductive statement, it can't remove all its values again - it gets "trapped". Formally, $u \models_{\mathcal{V}_{Trap}} I$ if and only if $\big(u\big) \cap \big(I\big) \neq \emptyset$.

Automaton 4.3: DFA for trap intepretation

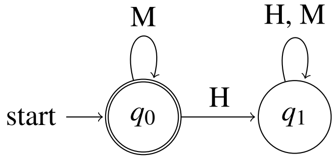
We denote with H all pairs in $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \in I$ *and M all pairs in* $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \notin I$.



Siphon Opposite to *trap*, *siphon* interpretation, a siphon I requires that none of its values is part of the configuration that satisfies I. Other words, $u \models_{\mathcal{V}_{siphon}} I$ if and only if $\big(u\big) \cap \big(I\big) = \emptyset$

Automaton 4.3: DFA for siphon intepretation

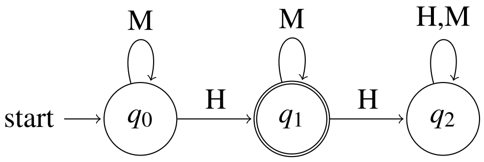
We denote with H all pairs in $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \in I$ *and M all pairs in* $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \notin I$.



Flow The third and last interpretation we are interested in is the flow interpretation $\mathcal{V}_{\{\updownarrow\wr\sqsupseteq}$. This time, we want that exactly at one position the letter of the configuration is part of the set in the same position in the encoded statement. Formally, $u \models_{\mathcal{V}_{siphon}} I$ if and only if $| \big(u\big) \cap \big(I\big) | = 1$

Automaton 4.3: DFA for flow intepretation

We denote with H all pairs in $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ such that $\sigma \in I$ and M all pairs in $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ such that $\sigma \notin I$.



Definition 4.4: Inductive statements

For any given Γ*-interpretation for* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$*, we define*

$$
\begin{aligned}
\mathit{Inductive}_{\mathcal{V}}(\mathcal{R}) &= \{I \in \Gamma^* | \forall u \rightsquigarrow_{\mathcal{T}} \ . \mathit{if} \langle u, I \rangle \in [[\mathcal{V}]] \ \mathit{then} \ \langle v, I \rangle \in [[\mathcal{V}]]\} \\
&= \{I \in \Gamma^* | \forall u \rightsquigarrow_{\mathcal{T}} \ . \mathit{if} \, u \models I \ \mathit{then} \ v \models I\}
\end{aligned}
$$

*Note that, for any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and any interpretation $\mathcal{V}$, any inductive statement $I \in \mathit{Inductive}_{\mathcal{V}}(\mathcal{R})$ that is satisfied in one configuration w ($w \models I$}) is also satisfied in all configurations that can be reached from w ($u \models I$ for all $w \rightsquigarrow^*_{\mathcal{T}} u$). *

Recall the example of *Token passing* system, we argue that $\emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^* \subseteq \mathit{Inductive}_{\mathcal{V}}(\mathcal{R})$. For the fixed length of configuration n = 4, all possible inductive statements are

$$\{n\} \ \ \{n\} \ \ \emptyset \ \ \emptyset$$

$$\{n\} \ \ \emptyset \ \ \{n\} \ \ \emptyset$$

$$\{n\} \ \ \emptyset \ \ \emptyset \ \ \{n\}$$

It concludes that *n n n t* can be reached from *t n n n* because all statements are satisfied by *t n n n* and also *n n n t* as well. But *t t n n* can not be reached from *t n n n* because $\{n\} \ \{n\} \ \emptyset \ \emptyset$ is not satisfied by *t t n n*.

By this way, we can guarantee that no bad configuations can be reached by checking both origin and the target configuations satisfied all the inductive statements.

Definition 4.5: Potential reachability

Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be any RTS and $\langle \Gamma, \mathcal{V} \rangle$ any interpretation. We write $u \Rightarrow_{\mathcal{V}} v$ if and only if $u \models_{\mathcal{V}} v$ for all $I \in \mathit{target}_{\mathcal{V}}(u) \bigcap \mathit{Inductive}_{\mathcal{V}}(\mathcal{R})$.

We will this definition for later for implemetation.

Lemma 4.1: *[Lam94]*

Let $\mathcal{R} = (\Sigma, I, T)$ be an RTS, $\langle \Gamma, \mathcal{V} \rangle$ an interpretation, and S a NFA over the alphabet Γ. Then there exists a $\Sigma - \Sigma -$ transducer C such that

$$[[C]] = \left\{ \langle u, v \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Sigma^n \mid \forall I \in \mathcal{L}(S)\, .\, if\ u \models_{\mathcal{V}} I\ then\ v \models_{\mathcal{V}} I \right\}$$

5 Algorithmic Learning of Finite Automata

Learning automata is a computational model for solving problems, where an agent learns to optimize its behavior by interacting with an unknown environment. The agent, also known as a learner, observes the feedback from the teacher, updates its internal state, and adjusts its actions accordingly. This interaction process between the *Learner* and the *Teacher* is the primary mechanism of learning automata. In the field of automata learning, there are generally two distinct settings: active and passive learning. Passive algorithms are provided with a fixed set of examples consisting of strings that the automaton should either accept or reject. Active algorithms, unlike passive ones, have the ability to expand the set of examples as needed by asking further queries. However, in this thesis, our focus is solely on active learning. We do not introduce passive learning here but refer the interested reader to [CES09].

This chapter aims to provide a deeper understanding of the process of learning automata, including the roles and responsibilities of the *Teacher* and *Learner* in Section 5.1. In Section 5.2, we will introduce several of active algorithms that use for our experiment.

5.1 The oracles

In this learning scenario, the *Teacher* is proficient in the language being taught and is responsible for answering any questions posed by the learner. The *Learner* is given the opportunity to ask two types of queries - membership and equivalence. Membership queries are used to classify a word based on whether it belongs to the language being taught or not. Equivalence queries, on the other hand, are used to determine whether an assumed automaton is equivalent to the language the *Teacher* has in mind. The learning process continues until the *Teacher* answers an equivalence query positively.

Membership oracle The *Learner* provides a word $w \in \Sigma^*$, the *Teacher* replies "yes" or "no" depending on whether $w \in \mathcal{L}$ or not.

Equivalent oracle The *Learner* conjectures a regular language, typically given as a DFA $\mathcal{M}$, and the *Teacher* checks whether $\mathcal{M}$ is an equivalent description of the target language $\mathcal{L}$

and return "yes", otherwise return an counterexample $u \in \Sigma^*$ with $u \in \mathcal{L}(\mathcal{M}) \iff u \notin \mathcal{L}$ or $u \in \mathcal{L} \iff u \notin \mathcal{L}(\mathcal{M})$.

On equivalent oracle, the *Teacher* can return a positive counterexample or a negative counterexample [Che+17]. A positive counterexample is a missing word in the conjecture but present in the target. The negative one is defined symmetic.

It is crucial for the *Teacher* to have a clear and specific understanding of the correct hypothesis. Since we know how to implement the *Teacher* to answer the oracles, it is now simple to apply different of learning algorithms.

5.2 Algorithms

A learning algorithm—often called learner—learns a regular target language $\mathcal{L} \subset \Sigma^*$ over an a priori fixed alphabet Σ by actively querying a teacher. We apply several of these algorithms in the course of this thesis.

5.2.1 L*

L* learning automata was introduced by Angluin in 1987 [Ang87], also called Angluin's algorithm. Angluin's algorithm has the ability to learn a regular set which is unknown initially, from any *Teachers*. During the learning process, it stores information in an observation table $O = (S, E, T)$ where $S \subseteq \Sigma^*$ is a nonempty *prefix-closed* [1] set, a finite *suffix-closed* [2] set E, and T : $(S \cup S \cdot A) \cdot E \to \{0, 1\}$ is a mapping that stores the table entries. The algorithm maintains $T(u) = 1$ if and only if u is accepted by the target language for all $u \in (S \cup S \cdot A) \cdot E$.

Let's take a closer look at the inner workings of the *Angluin's algorithm*. For each $row(s)$ of the table, where $s \in S$ denotes a function

$$f_s : E \to \{0, 1\} \; with \; f_s(e) = T(s \cdot e)$$

The overvation table has two properties: *closed* and *consistent*. An observation table is called *closed* provided that for each t in $S \cdot A$ there exists an s in S such that row(t) = row(s). An observation table is called *consistent* provided that whenever s_1 and s_2 are elements of S such that row(s_1) = row(s_2) for all a in A, row($s_1 \cdot a$) = row($s_2 \cdot a$). Once the table is *closed* and *consistent*, we can build a deterministic finite-state acceptor, which also is called *conjecture*, by using the

[1] A set of strings S is called prefix-closed if: $uv \in S \implies u \in S$

[2] A set of strings S is called suffix-closed if: $uv \in S \implies v \in S$

observation table. More precisely, *Angluin's algorithm* constructs the DFA $\mathcal{H} = (Q, q_0, \Sigma, \delta, F)$ where:

$$Q = \{row(s) : s \in S\},$$

$$q_0 = row(\lambda),$$

$$F = \{row(s) : s \in S \ \ and \ T(s) = 1\},$$

$$\delta(row(s), a) = row(s \cdot a).$$

Basically these two conditions *closed* and *consistent* guarantee that the transitions is well-defined. The observation table is *closed* ensures that every row in the lower part also occurs in the uper part. In other words, the row labeled by elements of S are the candidates of states of the automaton. *Consistent* condition implies that both words lead to the same state in the automaton, as they cannot be distinguished by any $a \in \Sigma^*$.

The peseudocode 1 presents Algluin's algorithm in pseudocode. Essentialy, in the begin of learning process, the algorithm guarantees that the table are *closed* and *consistent* by repeatly modifiding the columns and also the rows of the table. After every extension of the table, the algorithm fill the table by asking the membership queries for all table entries $u \in (R \cup R \cdot \Sigma) \cdot S$ for which no membership is yet present by asking the *Teacher* the membership queries. If the *Teacher* replies "yes", then set $T(u) = 1$, otherwise $T(u) = 0$. Once this is the case, the observation table satifies the conditions, Angluin's algorithm constructs a conjecture, which it submits to an equivalence query. The learning terminates once the teacher replies “yes” on an equivalence query. However, if the Teacher returns a new counterexample $t \in \Sigma^*$ the algorithm modifies the table by adding t and its prefixes to S and repeats the process by going to line 1.

5.2.2 NL*

In general, a nondeterministic finite automata *NFA* is often preferable to a deterministic finite automata *DFA* due to potentially exponential differences in their sizes (REFERENCE FOR COMPARISON OF NFA AND DFA). Therefore, learning algorithms for nondeterministic finite automata (NFA) are required. In this section, we will introduce another active learning algorithm called the *NL** algorithm [Bol+09], based on *L**. The *NL** concludes a residual finite-state automata (RFSA), a subclass of nondeterministic finite automata was introduced in the seminar work [DLT01].

Technically, it is possible to learn an RFSA instead of a DFA by modifying Angluin's algorithm L^* observation table. The proposed method involves selecting *prime rows*[3] as representations

[3] *prime row, RFSA-closed, RFSA-consitency* are defined in [Bol+09]

Algorithm 1 Algluin's learning algorithm [Ang87]

```
Input: A teacher for a regular language $L \subseteq \Sigma^*$
Initialize the observation table (S, E, T)
Ask membership queries for $\lambda$ and each $a \in \Sigma$
Repeat:
while (S,E,T) is not closed or not consistent do
    if (S,E,T) is not consistent then
        find $s_1$ and $s_2$ in S, and $e \in E$ such that
        $row(s_1) = row(s_2)$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$,
        add $a \cdot e$ to E,
        conducts membership queries.
    end if
    if (S,E,T) is not closed then
        find $s_1$ and $a \in \Sigma$ such that
        $row(s_1 \cdot a)$ is different from $row(s)$ for all $s \in S$,
        add $s_1 \cdot a$ to S,
        conducts membership queries.
    end if
end while
Once (S, E, T) is closed and consistent, make $M = M(S, E, T)$
if the Teacher replies with a counter-example t, then then
    add t and all its prefixes to S
    conducts membership queries.
end if
Util the Teacher replies "yes"
Terminate and return a conjecture $\mathcal{M}$
```

of the automaton's states, rather than utilizing *all rows* of the table. The proceed of the NL^* learning algorithm is mainly the same with L*. Similar to L^*, it is also repeatedly checked the *RFSA-closed*[3] and *RFSA-consitency*[3] properties, once the both properties are fullfill, it can contruct the conjecture and ask the equivalent query to the teacher.

5.2.3 Kearns-Vazirani

Another active learning algortihm is introduced in this thesis is *Kearns and Vazirani's* [KV94]. Unlike *Angluin's algorithm* it organizes its data in an ordered binary tree. It aims to minimize the number of membership queries by storing only one representative for each L-equivalence class in the tree. The data are stored in two non-empty set $R, S \subseteq \Sigma$, where R consists of *representatives* that are used to represent the equivalence classes of L. The set S includes *separating words* that are used to verify that two different representatives indeed represent different equivalent classes. More formally, *Kearns-Vazirani's* algortihm keeps a separating word $v \in S$ for any two representatives $u \neq u' \in R$ such that $uv \in L \Leftrightarrow uv \notin L$ is satisfied.

The organization of the binary tree is simple, while the inner nodes are labeled with the word of S, the leaf nodes are labled with words of R. The algorithm labels the root node's with $\epsilon \in S$. The main property is that for each subtree, it places on the subtree's root $v \in S$ and all the $u \in R$ depending on whether $uv \in L$ or not. When $uv \notin L$ u is put in the left subtree. Otherwise, $uv \in L$ u will be put in the right subtree. This procedure is recursively repeated at each subtree until all representatives are put in their own leaf node.

The conjecture of *Kearns-Vazirani's* algorithm is defined following: DFA $\mathcal{H} = (Q, \Sigma, q_o, \delta, F)$. Where the set of states $Q = R$. The final states F consist of all representatives $u \in R$ that are located in the right subtree of the root node. Since ϵ is always an element of R, the initial state $q_0 = \epsilon$.

5.2.4 Rivest-Schapire

The last algorithm we will introduce in this thesis is *Rivest and Schapire*. The different of this algortihm to *Angluin's algorithm* is, that uses a *reduced* version of Angluin's observation table that stores exactly one representative per L-equivalence class. The advantages are storing less data and asking less memberships queries. (In fact, this method has originally been introduced by Schapire).

5.3 Libalf: the Automata Learning Framework

**Libalf* is a comprehensive, open-source program library for learning finite automata. It was used for a large share of the experiments conducted in this thesis, and many of the algorithms used or developed in later chapters habe between integrated into the library.* It supports both for active and passive algorithms but in this thesis we only consider the active algorithms. The basis libraries that are required for libalf are libAmore and libAmore++ that support for representing the automaton such that NFA and DFA.

Internlly, *libalf* represents words as list symbols where each symbol is an integer data type. Thus, the maximal size of an alphabet is 2^{32} or 2^{64} depending on the architecture of the target machine.

6 Implementation

We use automata learning algorithms to solve regular model checking problems and generate inductive statements for a regular transition system.

6.1 Membership oracle

On a membership oracle, the learner provides a statement and asks the teacher if this statement whether inductive or not. As we described in Definition 4.4, a statement I is *inductive* if, for any transition $v \rightsquigarrow u$ where u satisfies I, u also satisfies the statement. Oone can implement the Membership Oracle by checking the acceptance of $\mathcal{M}$, where $\mathcal{M}$ is an automaton for $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ and negating the answer (Algorithm 2). The $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ is defined by:

$$\overline{Inductive_{\mathcal{V}}(\mathcal{R})} = \{I \in \Gamma^* \mid \exists u \rightsquigarrow_{\mathcal{T}} w \,.\, u \models I\, and\, w \not\models I\} \tag{6.1}$$

Let $\mathcal{T} = \langle P, \Sigma \times \Sigma, \Delta, p_0, E \rangle$ is a transducer and $\mathcal{V} = \langle Q, \Sigma \times \Gamma, \delta, q_0, F \rangle$ is an interpretation. The automaton of $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ is defined by $\langle Q \times P \times Q, \Gamma, \triangle, \langle q_o, p_0, q_o \rangle, E \times F \times (Q \setminus F) \rangle$ where

$$\triangle(\langle q_1, p, q_2 \rangle, I) = \exists \langle \sigma_1, \sigma_2 \rangle \in \Sigma \times \Sigma. \ \ (\delta(q_1, \langle \sigma_1, I \rangle),\ \Delta(p, \langle \sigma_1, \sigma_2 \rangle),\ \delta(q_2, \langle \sigma_2, I \rangle))$$

The states that are accepted by this automaton when each its parts are satified:

$$\begin{aligned} \delta(q_1, \langle \sigma_1, I \rangle) &\in F \\ \Delta(p, \langle \sigma_1, \sigma_2 \rangle) &\in E \\ \delta(q_2, \langle \sigma_2, I \rangle) &\notin F \end{aligned}$$

For every pair of initial word and its reached word through the transducer. Where the initial word is satified by the statement I, the reached word is not. From 6.1 it can guarantee that all statements, that are acepted by this automaton, are non-inductive.

Algorithm 2 Membership oracle

```
Input: Statement $\mathcal{I}$
Output: True or False
begin
  $\mathcal{M} \leftarrow getAutomaton(\overline{Inductive_{\mathcal{V}}(\mathcal{R})})$
  if $\mathcal{I} \in \mathcal{L}(\mathcal{M})$ then
      return false;
  else
      return true;
  end if
end
```

6.2 Equivalent oracle

Now the teacher receives a conjecture as input by learner, the teacher will check if the conjecture satifies the safety property and return *true* if yes. Otherwise, the learner receives a counterexample and repeats its process.

Firstly, we want to make sure that the automaton only accepts inductive statements. Again, we get the automaton of $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$, find the intersection with the hypothesis. If exists any non-inductive statement in the hypothesis, return it as counterexample.

Since hypothesis $\mathcal{H}$ does not accept any non-inductive statement, we will check with the safety problems to make sure that the hypothesis strong enough. Intuitively, the automaton $\mathcal{D}$ contains all pairs from initial and bad words, which is induced by the inductive statements $\mathcal{L}(\mathcal{H})$. In other words, the safety property is that the inductive statements should not induce the initial and bad word. We return true and terminates the algorithm if $\mathcal{L}(\mathcal{D}) = \emptyset$. Otherwise we obtain a counterexample $\langle u_1 \dots u_n,\ v_1 \dots v_n\rangle \in \mathcal{L}(\mathcal{D})$. From Lemma 4.1, we can see that $\mathcal{D}$ is intuitively the intersection of the automaton $[[C]]$ and $\mathcal{I} \circ \mathcal{B}$. Because regular languages are closed under complement, $[[\overline{C}]]$ is defined with:

$$[[\overline{C}]] = \left\{ \langle u, v\rangle \in \bigcup_{n\geq 0} \Sigma^n \times \Sigma^n \mid \exists I \in \mathcal{L}(S)\,.\, if\ u \models_{\mathcal{V}} I\ then\ v \not\models_{\mathcal{V}} I \right\} \tag{6.2}$$

Since computing $[[\overline{C}]]$ is more effectively, we will construct the automaton for $[[\overline{C}]]$ and complement it. Let $S = \langle P, \Gamma, \Delta, p_0, E\rangle$ is a transducer and $\mathcal{V} = \langle Q, \Sigma \times \Gamma, \delta, q_0, F\rangle$ is an interpretation. The automaton of $\overline{[[C]]}$ is defined by $\langle Q \times P \times Q, \Sigma \times \Sigma, \triangle, \langle q_o, p_0, q_o\rangle, E \times F \times (Q \setminus F)\rangle$ where

$$\triangle(\langle q_1, p, q_2\rangle, \langle \sigma_1, \sigma_2\rangle) = \exists I \in \Gamma.\ \ (\delta(q_1, \langle \sigma_1, I\rangle),\ \Delta(p, I),\ \delta(q_2, \langle \sigma_2, I\rangle))$$

Algorithm 3 Equivalent oracle

```
Input: Statement $\mathcal{I}$
Output: $True$, X, or $I \in \Gamma^*$
begin
    $\mathcal{M} \leftarrow getAutomaton(\overline{Inductive_{\mathcal{V}}(\mathcal{R})})$
    if $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{M}) \neq \emptyset$ then        ▷ Make sure that all statements are inductive
        return $I \in \mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{M})$
    end if
    $\mathcal{D} \leftarrow getAutomatonFor(\mathcal{L}(\mathcal{I}) \circ \overset{\mathcal{L}(\mathcal{H})}{\Rightarrow} \circ \mathcal{L}(\mathcal{B}))$        ▷ Check safety property
    if $\mathcal{D} = \emptyset$ then
        return True
    end if
    $\begin{bmatrix}u_1\\v_1\end{bmatrix} \dots \begin{bmatrix}u_n\\v_n\end{bmatrix} \leftarrow getWordFrom(\mathcal{L}(\mathcal{D}))$
    $I = disprove(\begin{bmatrix}u_1\\v_1\end{bmatrix} \dots \begin{bmatrix}u_n\\v_n\end{bmatrix})$
    if $I = null$ then
        return X        ▷ throw exception when can not disprove
    end if
    return I
end
```

The states that are accepted by this automaton when each its parts are satified:

$$\begin{aligned}\delta(q_1, \langle\sigma_1, I\rangle) &\in F\\ \Delta(p, I) &\in E\\ \delta(q_2, \langle\sigma_2, I\rangle) &\notin F\end{aligned}$$

Now we want to find a counterexample $I \in \mathit{Inductive}_{\mathcal{V}}(\mathcal{R})$ for the hypothesis, such that $u_1 \ldots u_n \models_v I$ and $v_1 \ldots v_n \not\models I$ to make sure that our inductive statements do not induce this pair any more. We can also call I an active counterexample since I is in the target language but was missing in the candicate language.

6.3 The word problem

It leads to the question whether $I \in \mathit{Inductive}_{\mathcal{V}}(\mathcal{R})$ exists such that $u_1 \ldots u_n \models_v I$ and $v_1 \ldots v_n \not\models I$. Because this problem (proved in [Lam94]) is in NP, it can be reduced, in polynomial time, to SAT (since SAT is NP-hard). Additionally, we show that Problem 3.1 can be solved in PTime for the interpretations V_{trap} and V_{siphon} .

Flow interpretation In this thesis we will use the CNF-SAT to extract separating inductive statements. This means that the entire formular is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals. We define for each pair $\langle\sigma, i\rangle$ where $\sigma \in \Sigma$ and $1 \leq i \leq n$ a literal. Intuitively, $\sigma \in I_i$ if and only if the model value of this literal is true. Firstly, we introduce the macro

$$\mathit{ExactlyOne}(V) = \bigvee_{v \in V} v \wedge \bigwedge_{v, v' \in V : v \neq v'} \neg(v \wedge v')$$

Recall that a statement is not inductive if there exists one transition $\left[\begin{smallmatrix}u_1\\v_1\end{smallmatrix}\right] \ldots \left[\begin{smallmatrix}u_n\\v_n\end{smallmatrix}\right]$ that is accepted by transducer $\mathcal{T}$ for which holds that $x_1 \ldots x_n \models I_1 \ldots I_n$ and $y_1 \ldots y_n \not\models_{\mathcal{V}_{flow}} I_1 \ldots I_n$. Formally, we add these clauses to the formular:

$$\mathit{ExactlyOne}\Big(\bigcup_{1 \leq i \leq n} \{\langle u_i, i\rangle\}\Big) \tag{6.3}$$

and

$$\neg\mathit{ExactlyOne}\Big(\bigcup_{1 \leq i \leq n} \{\langle v_i, i\rangle\}\Big) \tag{6.4}$$

The clause 6.3 guarantee that there is exactly $1 \leq i \leq n$ such that $x_i \in I_i$. The clause 6.4 guarantee that either there is no or more than one $1 \leq i \leq n$ such that $x_i \in I_i$. Semantically, we define a state

$\langle l, q, k\rangle \in \{0,1\} \times Q_{\mathcal{T}} \times \{0,1,2\}$ corresponds to the observation that one can reach the state q of $\mathcal{T}$ with a word $\left[\begin{smallmatrix} u_1 \\ v_1 \end{smallmatrix}\right] \ldots \left[\begin{smallmatrix} u_n \\ v_n \end{smallmatrix}\right]$ such that there are k many indices i where $x_i \in I_i$, on the other hand, there are l many indices j where $y_j \in I_j$. Now we make sure that for every pair $[x, y]$, which we consider, is accepted by the *transducer* $\mathcal{T}$. Futhermore, statement I at the final step should not induce the source and target configuration.

$$
\begin{gathered}
\bigvee_{q_0 \in Q_0^{\mathcal{T}}} \langle\langle 0, q_0, 0\rangle, 0\rangle \wedge \neg \bigvee_{f \in F_{\mathcal{T}}} \langle\langle 1, f, 0\rangle, n\rangle \vee \langle\langle 1, f, 2\rangle, n\rangle \\
\wedge \bigwedge_{1 \le i \le n,\ \left\langle q, \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right], p\right\rangle \in \Delta_{\mathcal{T}}}
\begin{pmatrix}
\langle\langle 0, q, 0\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle 1, p, 1\rangle, i+1\rangle \\
\wedge\langle\langle 0, q, 1\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle 1, p, 2\rangle, i+1\rangle \\
\wedge\langle\langle 0, q, 2\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle 1, p, 2\rangle, i+1\rangle \\
\wedge \bigwedge_{k \in \{0,1\}} \begin{pmatrix}
\langle\langle k, q, 0\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle k, p, 1\rangle, i+1\rangle \\
\wedge\langle\langle k, q, 1\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle k, p, 2\rangle, i+1\rangle \\
\wedge\langle\langle k, q, 2\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle k, p, 2\rangle, i+1\rangle
\end{pmatrix} \\
\wedge \bigwedge_{l \in \{0,1,2\}} \begin{pmatrix}
\langle\langle 0, q, l\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle 1, p, l\rangle, i+1\rangle \\
\wedge\langle\langle 0, q, l\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle 1, p, l\rangle, i+1\rangle
\end{pmatrix} \\
\wedge \bigwedge_{k \in \{0,1\}\ l \in \{0,1,2\}} \left(\langle\langle k, q, l\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle k, p, l\rangle, i+1\rangle\right)
\end{pmatrix}
\end{gathered}
\tag{6.5}
$$

Therefore, $I_i = \{\sigma \in \Sigma \,|\, \mathit{modelValue}(\langle\sigma, i\rangle) = \mathit{true}\}$.

SAT-Solver for trap and siphon interpretation Analogous to flow interpretation, we can also use SAT-Solver to find the separating inductive statement. We only consider for trap interpretation, the siphon is analog. In this case, we change the format of the state product to $Q_{\mathcal{T}} \times \{\mathit{true}, \mathit{false}\}$. Sequentially, the initial and final states are defined

$$
\bigvee_{q_0 \in Q_0^{\mathcal{T}}} \langle\langle q_0, \mathit{false}\rangle, 0\rangle \wedge \neg \bigvee_{f \in F_{\mathcal{T}}} \langle\langle f, \mathit{true}\rangle, n\rangle
$$

The transitions should be following:

$$
\wedge \bigwedge_{1 \le i \le n,\ \left\langle q, \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right], p\right\rangle \in \Delta_{\mathcal{T}}}
\begin{pmatrix}
\langle\langle q, \mathit{false}\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle p, \mathit{false}\rangle, i+1\rangle \\
\wedge\langle\langle q, \mathit{false}\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle p, \mathit{true}\rangle, i+1\rangle \\
\wedge\langle\langle q, \mathit{true}\rangle, i\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle p, \mathit{true}\rangle, i+1\rangle
\end{pmatrix}
\tag{6.6}
$$

A polynomial time algorithm for the word problem for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ Again, we focus on Vtrap since the arguments for Vsiphon are analogous. Pseudocode 4 show the method how we find the separating statement for trap interpretation in polynomial time. We start with the statment $I = \Sigma \backslash \{y_1\} \ldots \Sigma \backslash \{y_n\}$. If a transition $\left[\begin{smallmatrix}x_1\\y_1\end{smallmatrix}\right] \ldots \left[\begin{smallmatrix}x_n\\y_n\end{smallmatrix}\right]$ exists such that $x_1 \ldots x_n$ staifies the current statement and $y_1 \ldots y_n$ does not, the removes x_i from the i-th letter of the statement for all $1 \leq i \leq n$. To prove this approach can be computed in polynomial time, refers to [Lam94].

Algorithm 4 Disprove

Input: $\left[\begin{smallmatrix}x_1\\y_1\end{smallmatrix}\right] \ldots \left[\begin{smallmatrix}x_n\\y_n\end{smallmatrix}\right]$ and transducer $\mathcal{T}$

Output: Inductive statement I

```
begin
for $i = 1; i \leq n; i = i + 1$ do
    $I_i = \Sigma \backslash \{y_i\}$
end for
while $\langle v, I \rangle \in \mathcal{L}(\mathcal{V}_{trap})$ do
    if $\exists \left[\begin{smallmatrix}x_1\\y_1\end{smallmatrix}\right] \ldots \left[\begin{smallmatrix}x_n\\y_n\end{smallmatrix}\right]$ where $u_1 \ldots u_n \models I$ and $v_1 \ldots v_n \not\models I$ then
        for $i = 1; i \leq n; i = i + 1$ do
            $I_i = I_i \backslash \{u_i\}$
        end for
    else
        return I
    end if
end while
return $\emptyset$
end
```

*Note that the alphabet of the language that we learn is considerably large; i.e. exponentially larger than the alphabet of the RTS. *Libalf* supports starting a learning process with some alphabet which can be expanded later if necessary. Therefore, we start the learning process with an empty alphabet and gradually add those letter from 2^{Σ}. *

7 Experiments

7.1 Case studies

Dijkstra’s algorithm for mutual exclusion

Dijkstra’s algorithm for mutual exclusion with a token

Other mutual exclusion algorithms

Dining philosophers

Cache coherence protocols

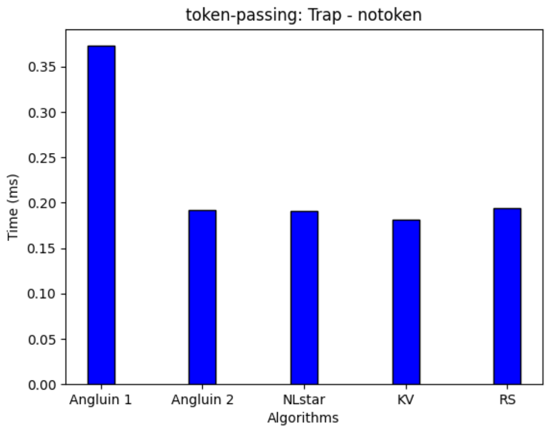
Termination detection

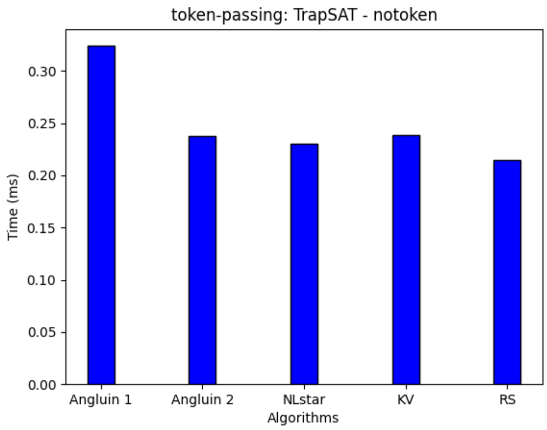
Dining cryptographers

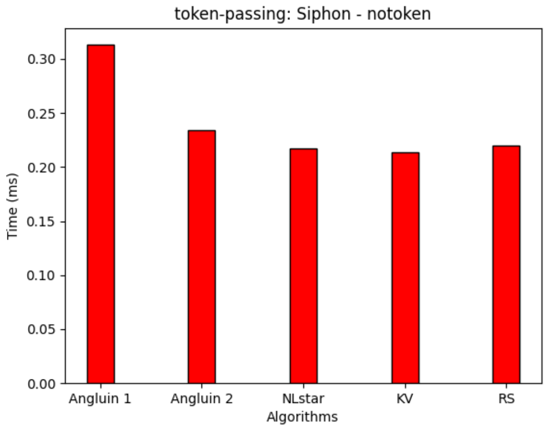
Leader election

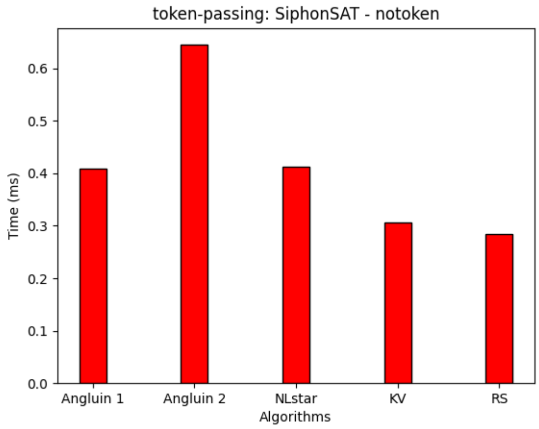
Token passing

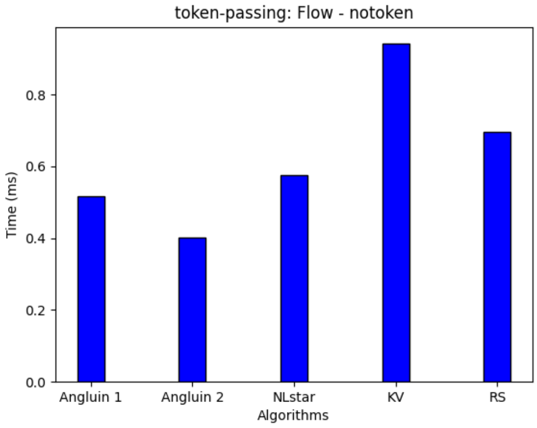
7.2 Results

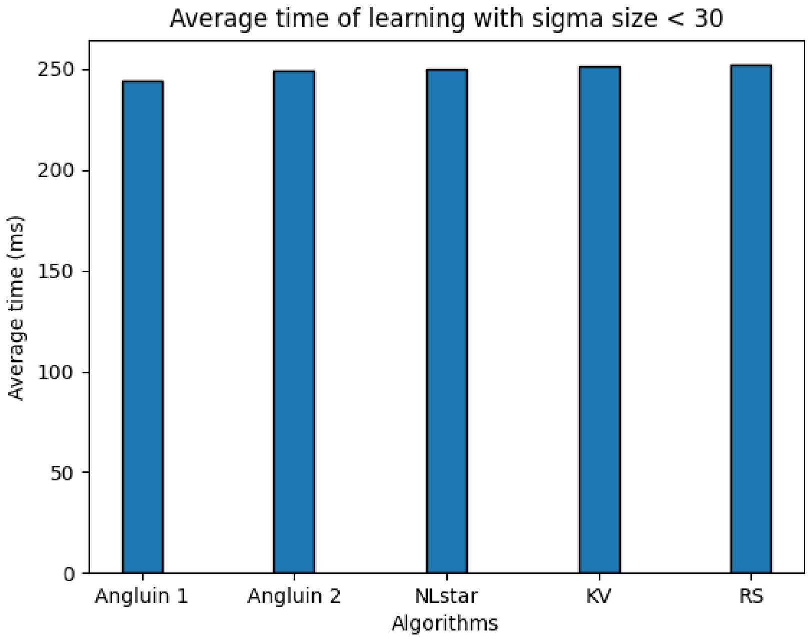

token-passing: Trap - notoken
Time (ms)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms

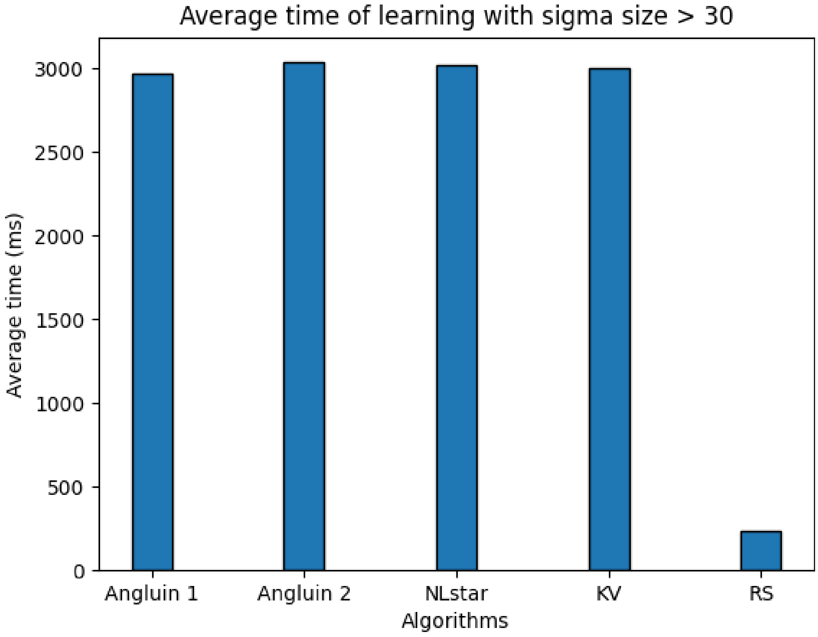

token-passing: TrapSAT - notoken
Time (ms)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms

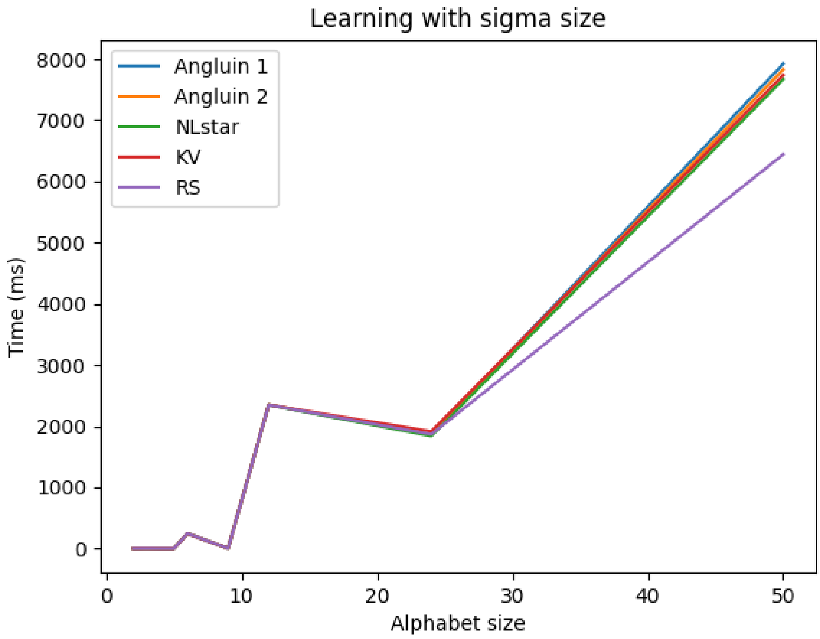

token-passing: Siphon - notoken
Time (ms)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms

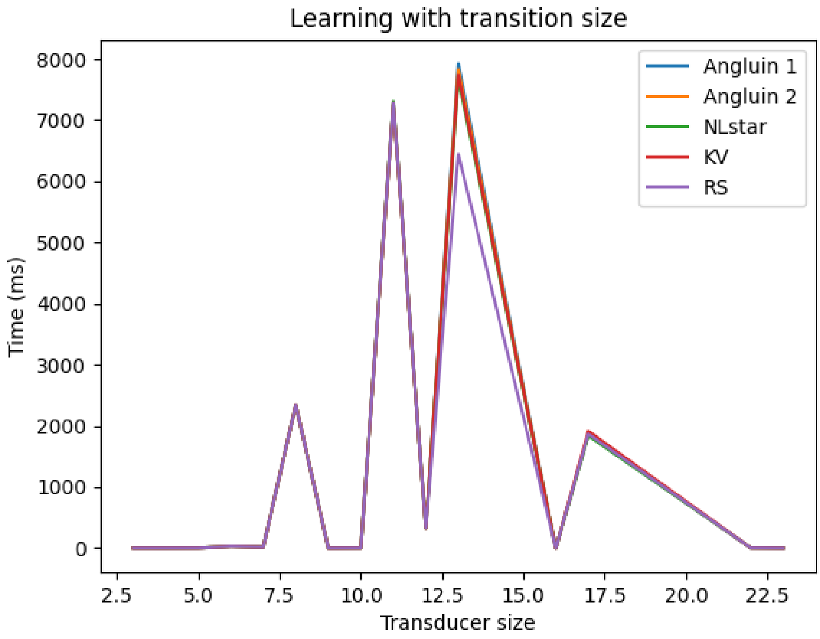

token-passing: SiphonSAT - notoken
Time (ms)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms

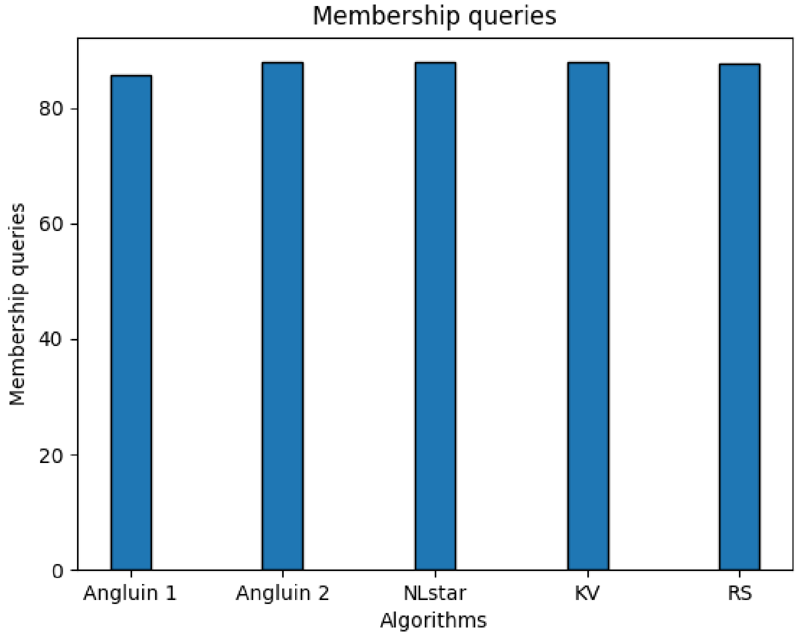

token-passing: Flow - notoken
Time (ms)
0.0
0.2
0.4
0.6
0.8
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms

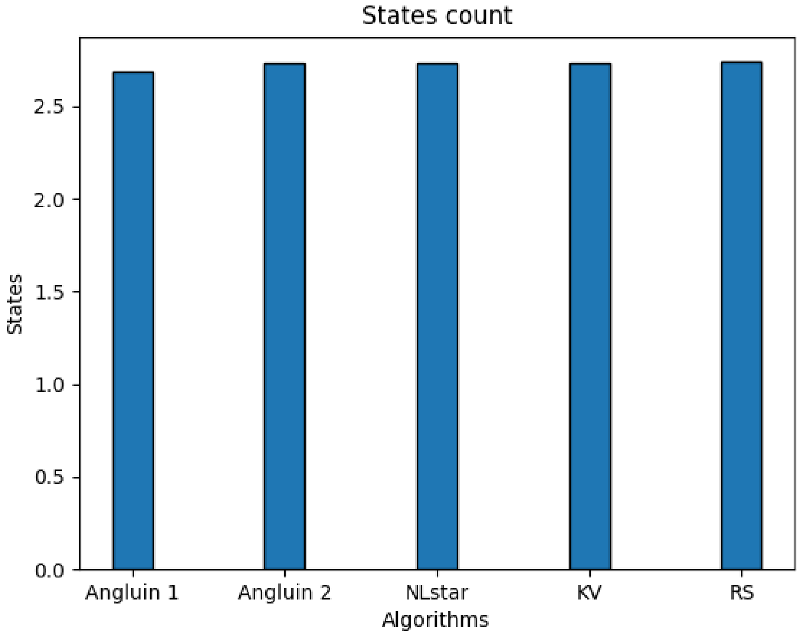

Average time of learning with sigma size < 30
Average time (ms)
0
50
100
150
200
250
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms


Average time of learning with sigma size > 30
Average time (ms)
0
500
1000
1500
2000
2500
3000
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms


Learning with sigma size
Angluin 1
Angluin 2
NLstar
KV
RS
Time (ms)
0
1000
2000
3000
4000
5000
6000
7000
8000
0
10
20
30
40
50
Alphabet size


Learning with transition size
Angluin 1
Angluin 2
NLstar
KV
RS
Time (ms)
0
1000
2000
3000
4000
5000
6000
7000
8000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
Transducer size


Membership queries
Membership queries
0
20
40
60
80
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms


States count
States
0.0
0.5
1.0
1.5
2.0
2.5
Angluin 1
Angluin 2
NLstar
KV
RS
Algorithms

8 Conclusion

We studied Regular Model Checking of safety properties. We evaluated the performance of our algorithms based on a prototype implementation.

Open Questions and Future Research

Abbreviations

List of Figures

List of Algorithms

List of Tables

Bibliography

[Ang87] D. Angluin. "Learning regular sets from queries and counterexamples." In: *Information and Computation* 75.2 (1987), pp. 87–106. ISSN: 0890-5401. DOI: `https://doi.org/10.1016/0890-5401(87)90052-6`.

[Bol+09] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. "Angluin-Style Learning of NFA." In: *International Joint Conference on Artificial Intelligence*. 2009.

[CES09] E. M. Clarke, E. A. Emerson, and J. Sifakis. "Model checking: algorithmic verification and debugging." In: *Communications of the ACM* 52.11 (2009), pp. 74–84.

[Che+17] Y.-F. Chen, C.-D. Hong, A. W. Lin, and P. Ruemmer. *Learning to Prove Safety over Parameterised Concurrent Systems (Full Version)*. 2017. arXiv: `1709.07139 [cs.LO]`.

[DLT01] F. Denis, A. Lemay, and A. Terlutte. "Residual Finite State Automata." In: *STACS 2001*. Ed. by A. Ferreira and H. Reichel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 144–157. ISBN: 978-3-540-44693-4.

[KV94] M. J. Kearns and U. Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[Lam94] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.